remote sensing

MDPI

Article

Maintaining Semantic Information across Generic 3D Model Editing Operations

Sidan Yao [1], Xiao Ling [1], Fiona Nueesch [1], Gerhard Schrotter [2], Simon Schubiger [3,*], Zheng Fang [4], Long Ma [4] and Zhen Tian [5]

1 ETH Zurich, Future Cities Laboratory, Singapore-ETH Centre, Singapore 138602, Singapore; yao@arch.ethz.ch (S.Y.); xlingsky@whu.edu.cn (X.L.); fiona.nueesch@sunrise.ch (F.N.)
2 Geomatics Engineering (GeoZ) City of Zurich, 8022 Zurich, Switzerland; gerhard.schrotter@zuerich.ch
3 Esri R&D Center Zurich, 8005 Zurich, Switzerland
4 Nanyang Technological University, School of Computer Science and Engineering, Singapore 639798, Singapore; zheng.fang@ntu.edu.sg (Z.F.); ma_long@ntu.edu.sg (L.M.)
5 Department of Computer Science & Engineering, Ohio State University, Columbus, OH 43210, USA; tian.393@osu.edu
* Correspondence: simon.schubiger@fhnw.ch

Received: 10 December 2019; Accepted: 15 January 2020; Published: 20 January 2020

Abstract: Many of today's data models for 3D applications, such as City Geography Markup Language (CityGML) or Industry Foundation Classes (IFC) encode rich semantic information in addition to the traditional geometry and materials representation. However, 3D editing techniques fall short of maintaining the semantic information across edit operations if they are not tailored to a specific data model. While semantic information is often lost during edit operations, geometry, UV mappings, and materials are usually maintained. This article presents a data model synchronization method that preserves semantic information across editing operation relying only on geometry, UV mappings, and materials. This enables easy integration of existing and future 3D editing techniques with rich data models. The method links the original data model to the edited geometry using point set registration, recovering the existing information based on spatial and UV search methods, and automatically labels the newly created geometry. An implementation of a Level of Detail 3 (LoD3) building editor for the Virtual Singapore project, based on interactive push-pull and procedural generation of façades, verified the method with 30 common editing tasks. The implementation synchronized changes in the 3D geometry with a CityGML data model and was applied to more than 100 test buildings.

Keywords: semantic building information; CityGML; building model; model matching; 3D information maintenance; automatic labeling; 3D user interface

1. Introduction

Many of today's data models for 3D applications, such as CityGML or IFC, encode rich semantic information in addition to the traditional geometry and materials representation. Semantic information enables a wide range of application scenarios beyond visualization. Specifically, semantic and hierarchical information of building models is important for building information analysis [1], urban analytics [2], deployment of facilities [3], prediction [4], and governance [5]. Semantic information provides the identification of key components in a generic 3D model and enables users to differentiate walls, roofs, ground, and other surfaces in a 3D building model. Hierarchical information carries the relationships between surfaces in 3D models; for instance, in 3D building models, it tells how a window is related to a wall surface. Nowadays, a variety of techniques are available for editing

generic 3D models. However, the maintenance of semantic and hierarchical information through the editing process is only supported by specific applications that are closely tied to the underlying data model. We identified five main classes of popular geometry editing techniques: transformation, deletion, deformation, push-pull [6,7], and parametric (procedural) templates [8–10]. Most editing tools preserve only geometric information, UV mappings, and materials. In order to provide a flexible integration of existing as well as future editing tools, an automatic matching and synchronization method for the modified geometry and the data model is needed. Many of today's 3D applications provide SDKs and APIs where 3D geometry, UVs, and materials are exposed, and thus enable an easy integration of the presented method.

There are three major challenges to maintaining semantic information across the editing process: the first one is to decide to what extent to depend on the specific edit operation. On the one hand, recovering information based on an edit operation is an intuitive way to maintain semantic information. On the other hand, this would mandate the same number of data model update strategies as edit operations, severely limiting its flexibility. Therefore, an automatic matching method independent of the specific edit operation is required. The second challenge is how to deal with both rigid and non-rigid transformations of polygons as well as how to distinguish existing faces from newly added faces. New faces need to be inserted and classified according to the data model. The third challenge is the performance of the automatic matching process, since it very likely runs after each interactive edit operation, and thus may impact the user experience by its running time.

The presented automatic matching method addresses all three challenges. It is based on a closed loop of interleaving interactive edit operations and automatic data model synchronizations. Starting from a source data model with geometry, UVs and materials, the method preserves semantic information after geometry changes through a process of registration, recovering, and labeling. The method is independent of the specific editing operation, and can thus be applied to a wide range of interactive 3D tools or 3D batch processing jobs. The main contributions of the method are:

- Independence of edit operation: The method assumes no knowledge about the edit operation. It is able to recover information only based on the source data model and the modified 3D geometry.
- Integration flexibility: Since the method is only based on 3D geometry, UVs and materials, which are usually exposed through SDKs and APIs, integration in existing 3D applications is simple.
- Information preservation: The method can preserve all of the semantic information of the original model, classify newly added polygons, and label them according to the data model.

The method was implemented and tested as part of a LoD3 building editor for the Virtual Singapore project using CityGML as the underlying data model. It features standard 3D transformation tools, push-pull tools, and procedural façade templates through the integration of 3rd party libraries. These libraries disregard the data model and operate on the 3D geometry only. The CityGML data model is updated automatically after each edit operation. Thirty editing test cases were designed to verify that the data model was consistent, and the semantic information was maintained after each edit operation.

2. Related Work

Several research areas are relevant for understanding the problem's context: generating and maintaining hierarchical and semantic information of 3D models (CityGML in particular), 3D model matching methods, and state-of-the-art 3D editing techniques.

2.1. Semantic Information

A semantically-enriched 3D scene is crucial for non-visualization oriented applications. Alegre [11] proposed a probabilistic approach to the semantic interpretation of building façades. Verdie [12] reconstructed a 3D urban scene by classifying and abstracting the original meshes. With the

assistance of Markov Random Field, Verdie's method is able to distinguish ground, trees, façades, and roof surfaces. Zhu [13] proposed a point cloud classification method based on multi-level semantic relationships, including point-homogeneity, supervoxel-adjacency, and class-knowledge constraints. Furthermore, Wu [14] presented an algorithm to produce hierarchical labeling of an RGB-D scene for robotics. Rook [15] automatically labeled faces using a decision tree but without support for LoD3 labels. All these methods address the broader problem of creating semantic information directly from 3D faces or point cloud data, and thus do not take into account prior information available in a rich source data model.

2.2. Model Registration

Model registration between the source geometry and the edited geometry allows recovering the transformation applied by an edit operation. Various methods were developed over the past to fully or partially match geometries and extract transformations between matching parts.

Sundar [16] proposed a skeleton based method for comparing and searching 3D models. The method expresses the model by a skeleton graph and uses graph searching techniques to match 3D models. DeepShape [17] extracts high-level shape features to match geometric deformations of shapes. A shape signature is computed with the method introduced in [16], which makes it possible to measure similarity of different shapes. This method represents the signature of an object as a shape distribution sampled from a shape function measuring global geometric properties of the object. Iterative closest point (ICP) is a widely used registration method introduced by Besl and McKay [18] and Zhang [19]. This efficient point set registration method meets well, the requirements in terms of interactive edit performance and simple geometry representation. ICP was proved to be adequate for the editing operations used in the test cases (see Table 1). If necessary, ICP variants such as [20] and [21] can be integrated. Other alternatives include: Coherent point drift(CPD), a probabilistic method introduced by Myronenko [22] for both rigid and non-rigid point set registration. Ma proposed his vector field consensus algorithm in [23]. Ge [24] presented the global-local topology preservation (GLTP) algorithm for non-rigid point set registration.

2.3. Editing Techniques

Push-pull (press-pull or sculpting) is a popular, intuitive editing technique which unifies previously distinct tools such as edge move, face move, face split, and extrude into a single tool. Having a single tool minimizes tool changes and thus improves the efficiency of the editing process. It has been adopted by many commercial products, such as AutoCAD [25], SketchUp [7], and Maya [26]. In particular, PushPull++ [6] used in CityEngine [27], and ArcGIS Pro [28] employs methods for adaptive face insertion, adjacent face updates, edge collapse handling, and an intuitive user interface that automatically proposes useful drag directions. PushPull++ has been verified to be able to reduce the complexity of common modeling tasks by up to an order of magnitude comparing to existing tools, and is therefore well suited for editing building models and is integrated as part of the Virtual Singapore Editor (VSE). Esri's procedural runtime [10] is used to generate parametrized geometry from façade templates. The procedural runtime is a library implementing the computer graphics architecture (CGA) shape grammar introduced in [8,9], which defines a set of operations for transforming shapes in space, with a special focus on patterns found in architecture.

2.4. CityGML

While the method is independent of the data model, the VSE is based on CityGML [29], introduced by Kolbe [30], covering the geometrical, topological, and semantic aspects of 3D city models. CityGML defines the classes and relations for relevant topographic objects in cities and regional models with respect to their geometrical, topological, semantical, and appearance properties. There is wide industry support for importing CityGML, (e.g. Bentley Map [31], BS Contact Geo [32],

CityServer3D [33], CodeSynthesis XSD [34], FME from Safe Software [35], and many more), but editing of the data model is supported by only a few options (e.g. [36,37]).

2.5. Information Synchronization

The method regards edit operations as black boxes, but looking at current tools, it helps to distinguish five main types of editing techniques: transformation, deletion, deformation, push-pull, and template applications.

The following section gives an overview of the method and presents each step. Then, the key tasks of registration, recovery, information transfer between source and edited model, and automatic labeling are discussed.

Internally, 3D geometry is represented as face-vertex meshes [38] which incorporate a list of vertices and a set of faces referencing their vertices. Both the source geometry and edited geometry are handled as meshes (M_s and M_e). Mesh M_s is represented by vertices $v \in V$ which are points in three-dimensional space. A vertex v_i may optionally be annotated by a two-dimensional texture coordinate $t_i \in T$. For each pair of connected vertices in f, an edge $e = (v_i, v_j)$ is defined. Planar faces $f \in F$ are defined by a list of counter clockwise oriented vertices $(v_1, v_2, \ldots, v_n)$. Opening (holes) are represented as faces referencing their boundary (parent) faces.

3. Method Overview

Figure 1 gives an overview of the model synchronization method. Applying an edit operation to a source geometry (M_s) results in an edited geometry (M_e) deprived of semantic information because we assume that these are not preserved by the edit operation.

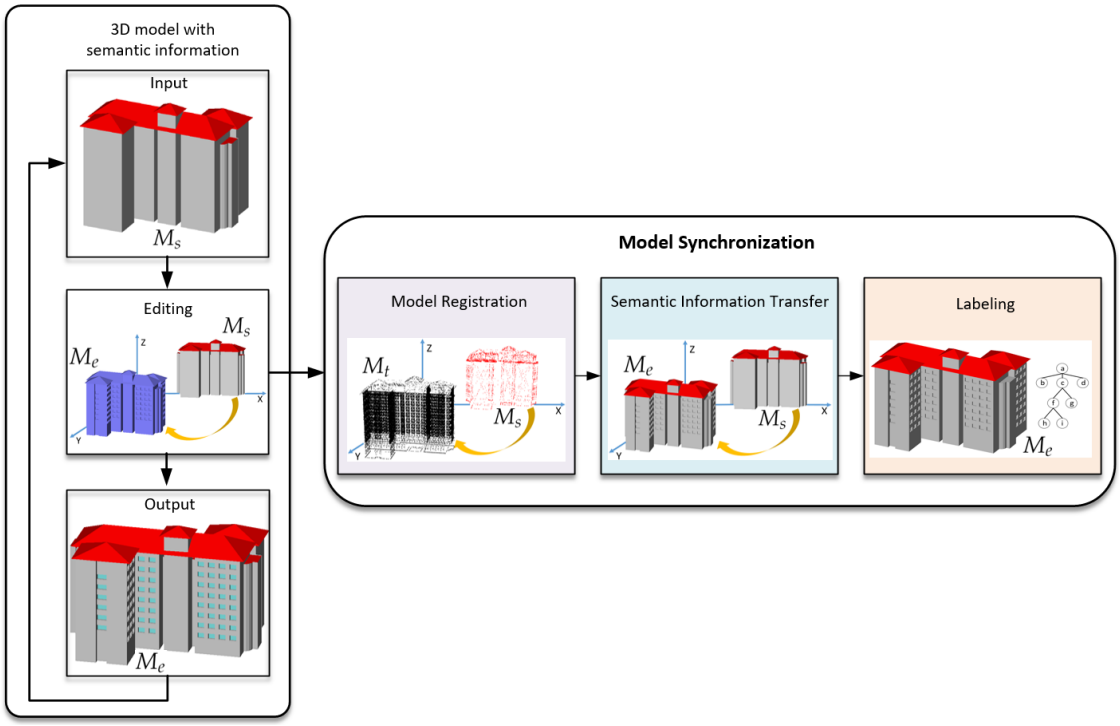

Figure 1. Flow chart of automatic 3D model synchronization.

The model synchronization method can be divided into three steps:

1. **Model registration**: M_s is the source point set for the ICP algorithm, whereas M_e is its target point set. The output of ICP is a transformation matrix which is applied to the source M_s to yield a transformed point set M_t. An octree storing additional information (face normal, face centroid, length of edges) is constructed from the transformed model M_t.

2. **Semantic information transfer**: An octree search with faces from M_e yields matching rigid transformed faces in M_t and their original semantic labels in M_s. Deformed faces are handled in a similar way by an additional UV space search.
3. **Labeling**: Each face in the edited geometry M_e is incorporated into a topological tree to detect openings and their boundary surfaces. Boundary surfaces are labeled as roof, wall, or ground faces based on their normal direction. For openings, the octree and UV space are used again to find the position in the data model, finally resulting in an updated data model that preserves as much semantic information as possible and which is updated with new faces labeled according to their topology.

3.1. Face Matching

Since M_e contains only geometric information after an edit operation, faces $f_e \in M_e$ have to be matched with corresponding faces $f_s \in M_s$ to find related semantic information. The following feature are used for face matching:

- Face normal f_{normal}.
- Face centroid $f_{centroid}$.
- Length of edges f_{length}.
- Set of texture coordinates f_T.

If the distance between two vertices v_i and v_j is below a given threshold ϵ_v, they are considered the same. Likewise, if the UV distance of two texture coordinates t_i and t_j is below a given threshold ϵ_t, their UVs are considered the same (See Figure 2).

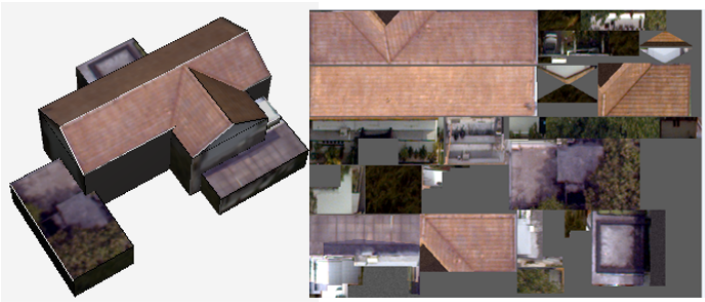

Figure 2. UV space of a building model. In UV space, each vertex v_i of the geometry has a corresponding texture coordinate t_i. The UV space simplifies matching faces, since texture coordinates remain unchanged or are re-sampled during deformation, push-pull, and template applications.

Two faces $f_t \in M_t$ and $f_e \in M_e$ are matched if:

- f_t and f_e have same f_{normal}, $f_{centroid}$, and f_{length}.
- f_t and f_e share same set of texture coordinates.

For matching faces, semantic information from $f_s \in M_s$ can be transferred to $f_e \in M_e$ via $f_t \in M_t$.

3.2. Model Registration

Several editing operations include rigid transformation and scaling, which can be represented by a transformation matrix. The ICP algorithm is used by the method to calculate the transformation matrix between the source M_s and the edited geometry M_e. ICP can be applied to the source geometry as a whole to compute the global transformation of the geometry or partially to recover local transformations.

3.2.1. ICP Algorithm

The ICP algorithm is a simple and efficient way to find the best match between two point sets by solving an optimization problem. The inputs of the ICP algorithm are two point sets, the source point set X (see Equation (1)), and target point set P (see Equation (2)); the point sets may have different sizes, and the distance between two points is defined by an euclidean metric.

$$X = \{x_1, \ldots, x_{N_x}\} \tag{1}$$

$$P = \{p_1, \ldots, p_{N_p}\}. \tag{2}$$

The ICP algorithm finds the transformation T in an iterative approach, applying transformation T to X and computing the fitness error $E(T)$, until $E(T)$ is less than a given threshold.

$$E(T) = \frac{1}{N_p} \sum_{i=1}^{N_p} ||x_i - T(p_i)||^2. \tag{3}$$

3.2.2. Point Set Generation

In order to increase the quality of the ICP result for a given edit operation, more points than just the vertices of the source geometry are usually required. Generating points by sampling uniformly the edges is easy to implement but will result in the same points for symmetries, and thus prevent ICP from recovering, e.g., rotation or mirroring along these symmetries (see Figure 3). Random sampling is a widely used strategy [39,40], but may add an error to the ICP result because of the noise introduced by the random sampling.

Figure 3. A building model with symmetries. Using a uniform sampling strategy will prevent ICP from recovering a rotation by 180°.

Our method uses random sampling along edges and diagonals but with a consistent random sequence based on a random seed computed from the texture coordinate t_i of vertex v_i. As illustrated in Figure 4, additional points for the ICP algorithm are generated along diagonals by connecting opposite vertices. If the source geometry does not have texture coordinates, they are generated beforehand. Initializing the random number generator from the texture coordinates results in a unique sequence of sampling points for every edge and diagonal in M_s, and hence in M_e, which then can be unambiguously registered by the ICP algorithm.

The edge and diagonal sampling is done as follows:

For a segment S_{ij} with vertices (v_i,v_j) and texture coordinates (t_i,t_j):

1. Calculate two random seeds $seed_i$ and $seed_j$ based on the texture coordinates of t_i and t_j.
2. Generate n random numbers $r_{i_1}, r_{i_2}, \ldots, r_{i_n}$ based on $seed_i$, and n random numbers $r_{j_1}, r_{j_2}, \ldots, r_{j_n}$ based on $seed_j$. By applying these random numbers, we obtain n sampling points s_{ij_k} and n sampling points s_{ji_k} (see Equation (4)).
3. Add the points obtained by the second step to the source point set and target point set of ICP algorithm.

$$s_{ij_k} = v_i + r_{i_k}(v_j - v_i)$$
$$s_{ji_k} = v_j + r_{j_k}(v_i - v_j). \quad (4)$$

This enables the ICP algorithm to register the source geometry to the edited geometry, since there exist unique corresponding point sets for each segment in M_s and M_e (See Figure 5).

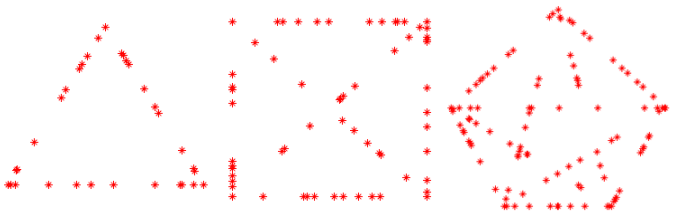

Figure 4. Edges and diagonals of a face are sampled with a random number generator which is initialized by the texture coordinates of the vertices of the edge or diagonal.

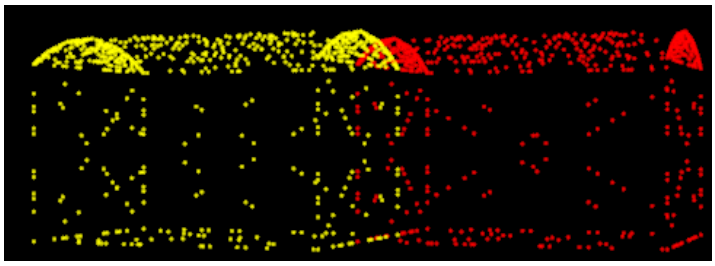

Figure 5. Example of a generated source point set (red) and generated target point set (yellow) for the iterative closest point (ICP) algorithm.

3.2.3. Parameter Analysis

Two parameters in ICP need to be determined for our method. First, the fitness error between M_s and M_e; 0.01 m is sufficient for buildings reconstructed from airborne and ground-based point clouds. Second, the maximum number of iterations.

Considering that VSE supports five main types of editing operations, six experiments were designed to derive the value of maximum number of iterations:

1. Translate 1000 m;
2. Translate 10 m;
3. Deletion on a face;
4. Deformation on a face;
5. Push-pull on a face;
6. Template application on a face.

These experiments were executed on a complete building model with 495 faces (see Figure 6). Figure 7 shows source point set and target point set.

Figure 6. A complete building model to derive the maximum number of iterations.

Figure 8 shows that deletion, deformation, push-pull, and template applications terminate ICP algorithm in the very beginning. For all experiments, fitness error falls below the threshold after no more than 90 iterations. In these experiments, maximum number of iterations can be set to an arbitrary value over 90. It was set to 300 to deal with potential complicated situations.

The registration result shown in Figure 9 indicates that 300 ICP alignment iterations and a fitness error of 0.01 m are sufficient to calculate the transformation matrix T. The red point set was generated from M_s, the yellow point set was generated from M_e, and the white point set was the transformed source point set by applying T to source point set.

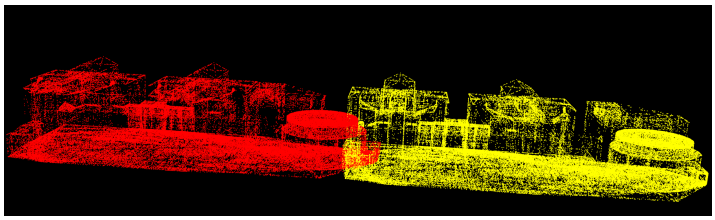

Figure 7. The source point set and target point set for the building model in Figure 6.

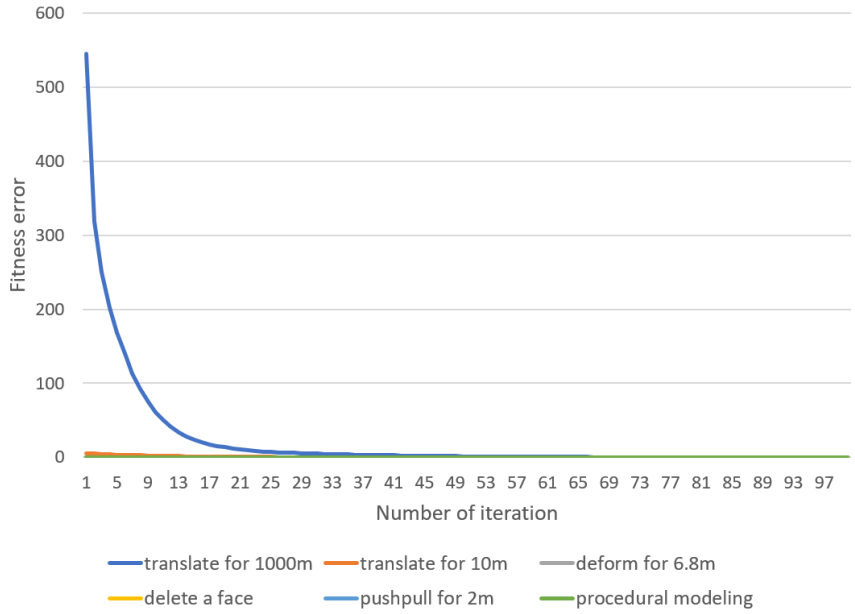

Figure 8. Experimental results under different numbers of iterations.

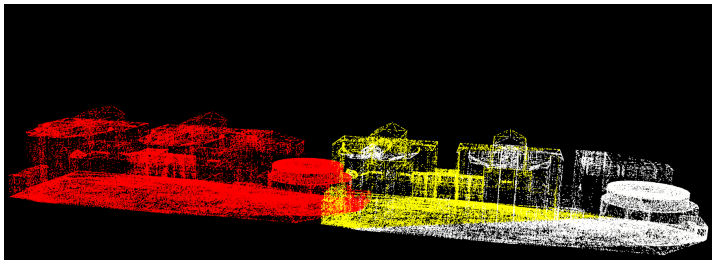

Figure 9. Registration result. The white point set is the transformed source point set. It overlaps with the point set generated from the edited geometry M_e.

3.3. Semantic Information Transfer

To match semantic information from the source model to the edited geometry, a search for matching faces had to be performed. The implementation uses an octree to speed up the search process.

An octree is a hierarchical data structure allowing efficient search operations in three-dimensional space. To match the semantic information of the original model, we constructed an octree based on the transformed source geometry $M_t = T \times M_s$. The root node of the octree is the bounding box of the source model. The bounding box is recursively divided into eight nodes until each box contains only one node as shown in Figure 10. Each node stores the face features from Section 3.1.

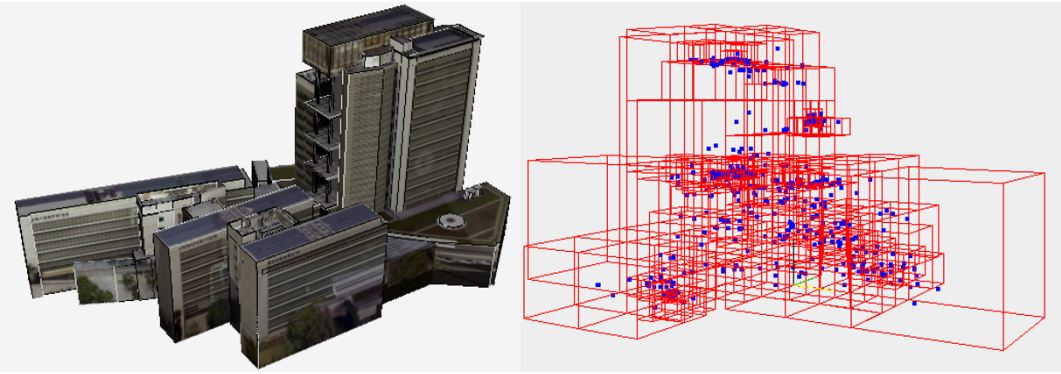

Figure 10. A building model and its octree.

Whether a face f_e in the edited geometry matches a face in the source data model is determined by searching for the corresponding face features (see Section 3.1) in the octree. Matching faces allow the transfer of semantic information from the source model M_s to the edited geometry M_e.

As illustrated in Figure 11, the deformation of a vertex, edge, or face changes f_{normal}, $f_{centroid}$, and f_{length}. Such a deformed face will not be found by an octree search. However, deforming edit operations only affect the geometry but preserve texture coordinates. Therefore, these faces can still be matched by an UV space search, a method also used for texture transfer (see for example [41]), and hence can be used to transfer semantic information.

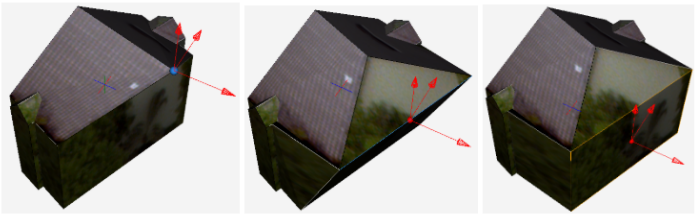

Figure 11. Deformation operations on vertex, edge, and face. Deformation will change f_{normal}, $f_{centroid}$ and f_{length}. A UV space search was performed for these faces.

After the octree and UV search, unmatched faces in the source data model M_s are deleted, and unmatched faces in the edited geometry M_e are considered new faces. For these, semantic information has to be generated through labeling, which will be discussed in the next section.

3.4. Labeling

Semantic information transfer based on faces can be applied as described to any data model. However, labeling new faces is domain specific and the approach described here is based on CityGML, which is the underlying data model of the implementation. According to the CityGML building model [42] the relevant classes for LoD3 building model are divided into three levels (see Figure 12), which are building, boundary surface, and opening. Specifically, the boundary surfaces are roof surface, wall surface, and ground surface, whereas the openings include windows and doors.

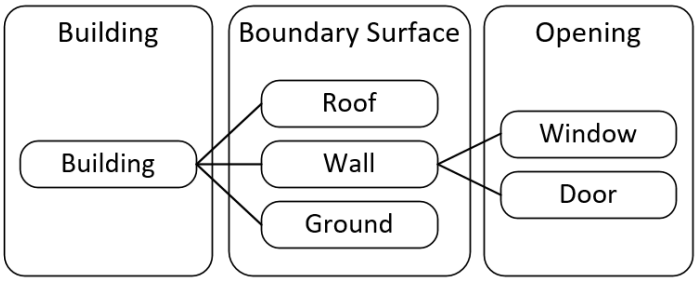

Figure 12. Relevant parts of the CityGML data model for LoD3 modelling.

Typical edit operations creating new boundary surfaces or openings are push-pull (see Figure 13) and template applications (see Figure 14). New boundary surfaces are classified according to the following heuristic where f_ϕ is the angle between the face normal f_{normal} and the up vector of the coordinate system, as shown in Figure 15. This basically assumes that walls are within $+/-15$ degree from the vertical (90 degree):

$$\begin{cases} f_\phi < 75^\circ, & \text{roof surface} \\ 75^\circ \leq f_\phi < 105^\circ, & \text{wall surface} \\ f_\phi \geq 105^\circ, & \text{ground surface} \end{cases} . \quad (5)$$

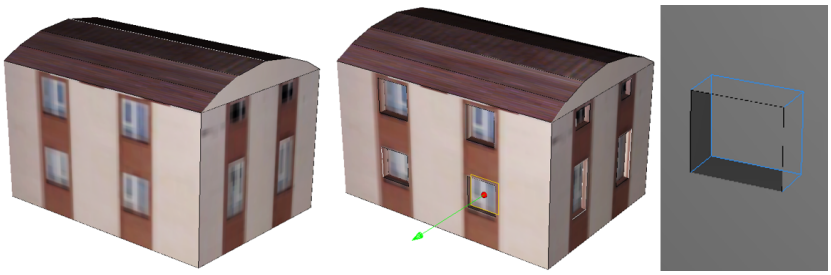

Figure 13. Opening generated by push-pull. This editing operation will not split the boundary surface but generate a hole on the boundary surface.

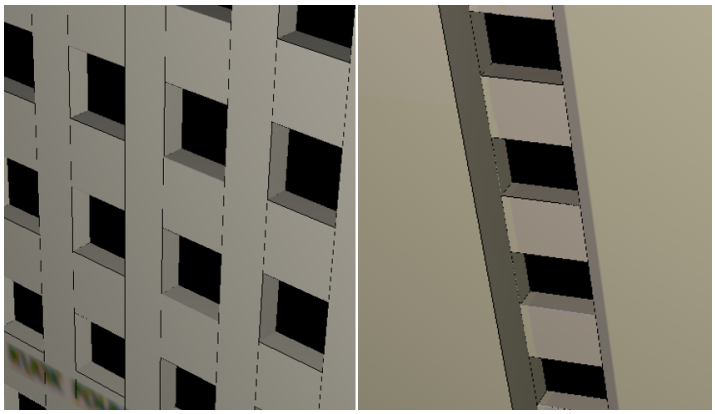

Figure 14. Openings generated by template application. This editing operation will split the boundary surface and generate several new faces co-planar with the boundary surface.

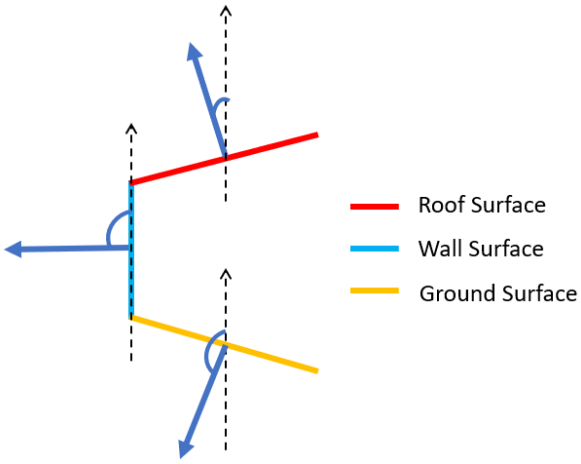

Figure 15. f_ϕ: the angle between the face normal f_normal and the up vector of the coordinate system.

Starting from a boundary surface (labeled either by semantic information transfer or by the heuristic), topological relationships are constructed by the iterative method depicted in Figure 16.

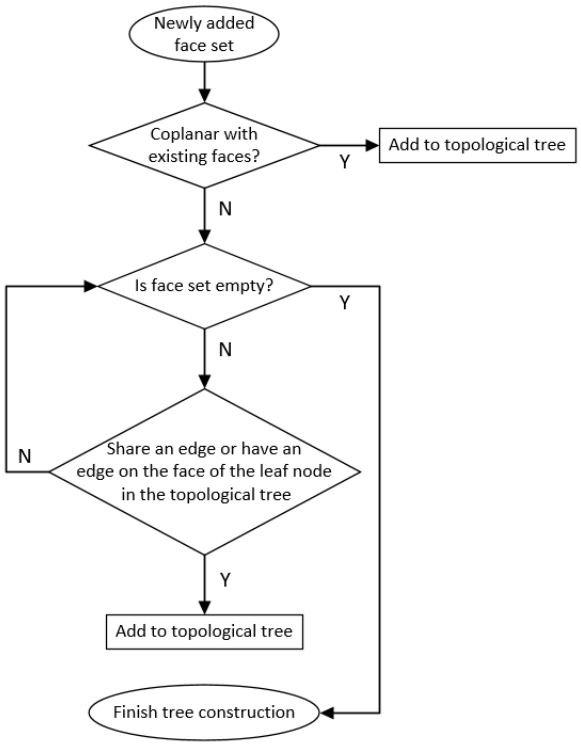

Figure 16. Openings detected by topology relationships.

This iterative approach constructs a topological tree based on the edited geometry. First, all new faces which are co-planar with labeled faces are collected and added to the topological tree as the first level. Second, it is determined whether the rest of the newly added faces *share* an edge or have an edge *on the face* of the leaf node in the topological tree. If yes, we add these faces to corresponding leaf nodes. This process is repeated until there are no new faces left. Finally, leaf nodes of all wall surfaces and with tree depth larger than two are considered as openings. Openings are classified as follows with h denoting the height and w denoting the width of the bounding box of the opening in meters. The constraints for doors are taken from [43].

$$\begin{cases} 2.0 < h < 2.5 \wedge 0.8 < w < 1.2, & \text{door} \\ \text{else}, & \text{window} \end{cases}.$$

In addition, roof infrastructures generated by push-pull operations (see Figure 17) are not openings, since in the topological tree they are children of roof surfaces.

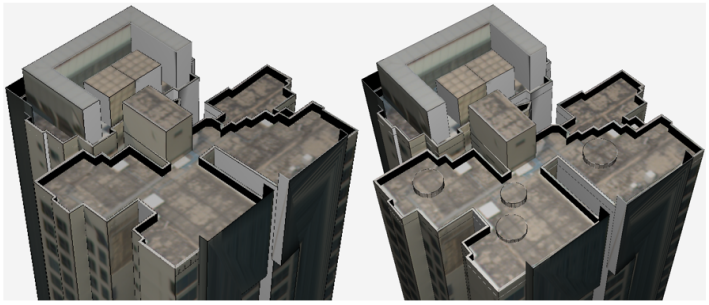

Figure 17. Adding roof infrastructure for building models by push-pull.

4. Discussion

The method was implemented as part of a standalone JavaFX based LoD3 building editor for the Virtual Singapore project. The Esri procedural runtime provides a façade template application and the PCL (Point Cloud Library [44]) and ICP implementations, along with other utilities for point set management. Besides camera control, CityGML import/export, and point cloud visualization, selection, and texture assignment, Figure 18 shows the user interface of VSE, the following editing operations are supported by the editor:

- Model move, scale, rotate (see Figure 9).
- Vertex, edge, and face move (see Figure 11).
- Face and edge push-pull (see Figure 13).
- Template application (see Figure 14).
- Face split (see Figure 19).
- Vertex, edge, and face deletion (see Figure 20).

We used FZKViewer [45] as 3rd party software to verify that the CityGML attributes were maintained when editing the building model. To illustrate typical edit operations and how the method updates the data model, two use cases are presented in the following sections: face deletion for shelters and template application for openings.

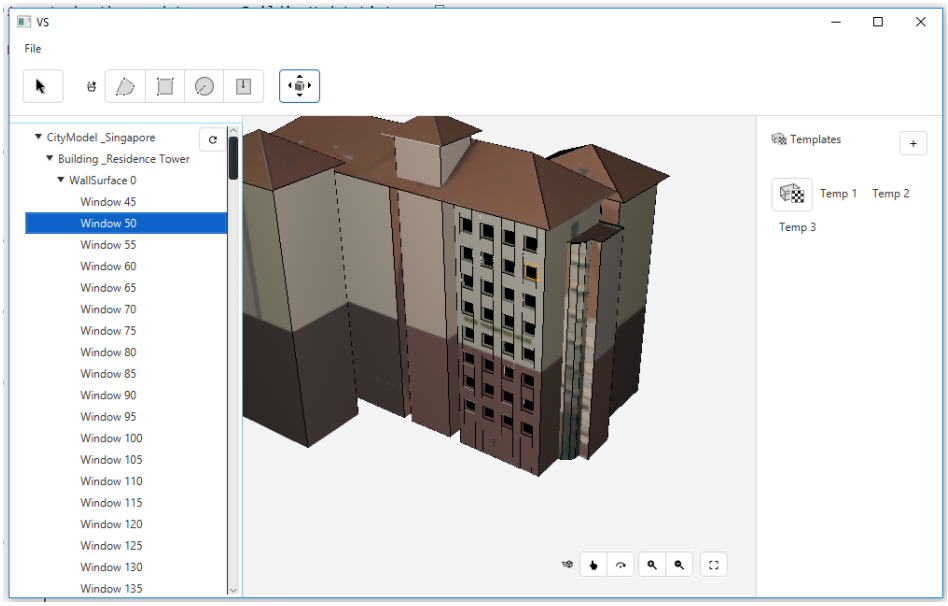

Figure 18. User interface of VSE.

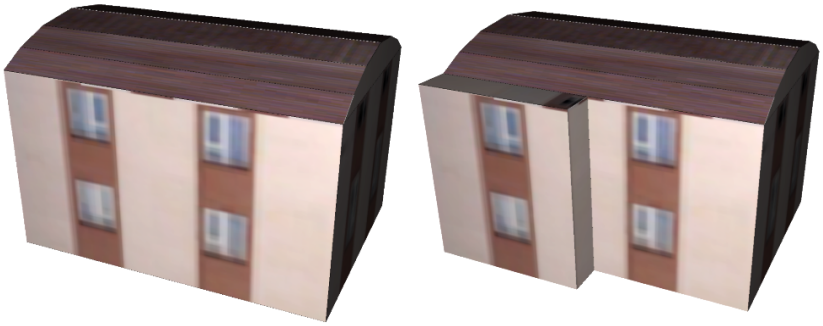

Figure 19. Face split operation.

4.1. Face Deletion For Shelters

A typical example for superfluous faces and incorrect semantic information are faces which are present because of occlusions at ground level. The passageway under the shelter, shown in Figure 20, was constructed as a volume instead of just the roof section of the passageway. The editor allows the user to remove superfluous faces through deletion and the data model is updates accordingly (see Figure 21).

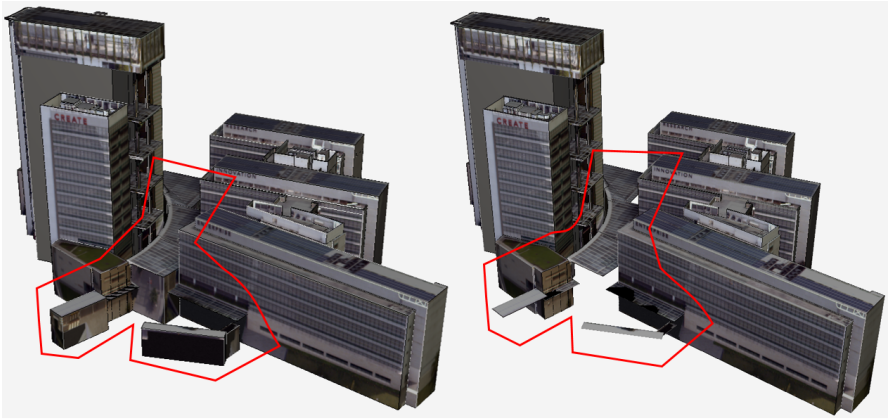

Figure 20. Deleting faces for shelters in the editor.

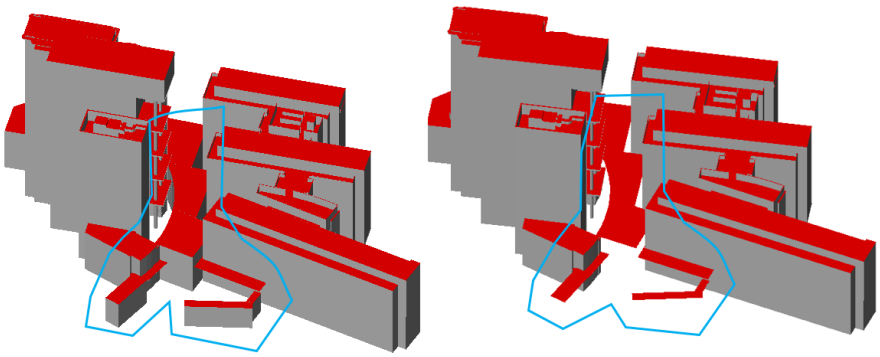

Figure 21. FZKViewer: Data model update after deletion. Roofs surfaces in red, wall surfaces in grey.

4.2. Template Application For Openings

Procedural modelling is used for adding regular structures; e.g., extending façade patterns into occluded areas or correcting measurement errors in the source model. Figure 22 shows hundreds of added openings in a high-rise building. Through the labeling process of the method, openings are correctly detected and classified as windows in the data model (see Figure 23).

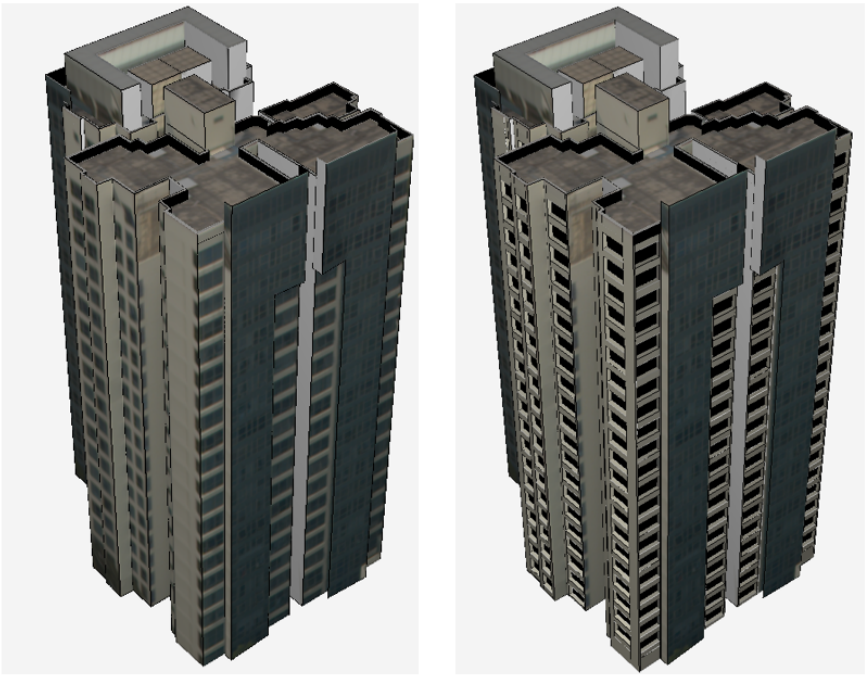

Figure 22. Façade template application.

These two use cases illustrate the automatic data model update process, which is essential for an effective editing workflow. While the increase in efficiency, compared to manual labeling, largely depends on the editing tasks, data model consistency has shown itself to be one of the greatest benefits of the method.

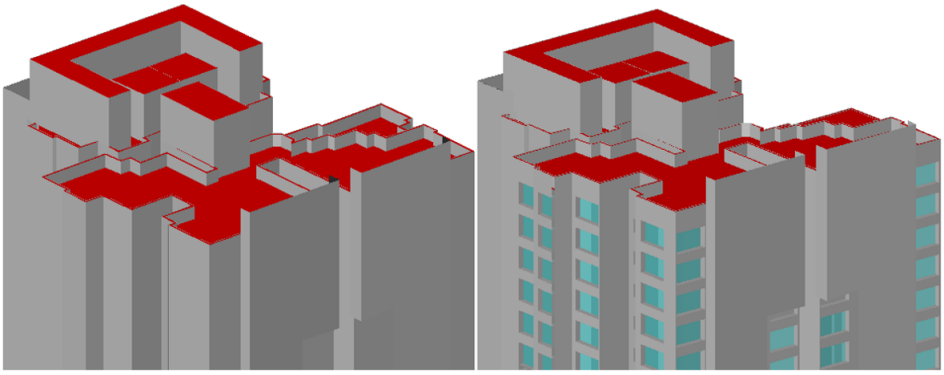

Figure 23. FZKViewer: Data model update for template application. Roofs surfaces in red, wall surfaces in grey, windows in turquoise.

Table 1. Test cases for verifying our model synchronization method.

Editing Type	Editing Operation	ID
Transformation	Translate the entire model	01
	Rotate the entire model	02
Deletion	Delete single vertex	03
	Delete multiple vertices	04
	Delete edge between two non-planar faces	05
	Delete edge between two co-planar faces	06
	Delete face with holes	07
	Delete face without holes	08
	Delete face with child faces	09
	Delete face without child faces	10
Deformation	Move single vertices	11
	Move multiple vertices	12
	Move single edge	13
	Move multiple edges	14
	Move single face	15
	Move multiple faces	16
	Move arbitrary selection of faces/edges/vertices	17
Push-pull	Push-pull of existing face follow the face normal	18
	Push-pull on existing face with an angle	19
	Push-pull on existing face in corner vertically	20
	Push-pull on existing face in corner with an angle	21
	Push-pull on existing edge	22
	Push-pull after creation of a face on an existing face	23
	Push-pull after creation of a edge on an existing face	24
	Push-pull after creation of multiple face on an existing face	25
	Push-pull after creation of multiple face on multiple existing face	26
	Push-pull after creation of a face on an existing face with holes	27
	Push-pull after creation of a circle on an existing face	28
Template	Apply procedural templates with one level of opening extrusion	29
	Apply procedural templates with two levels of opening extrusion	30

4.3. Commercial Tools

Several commercial tools are available which support editing of CityGML models. The following sections discuss a selection of these tools in terms of their advantages and limitations regarding editing of LoD3 CityGML building models.

4.3.1. CityEditor

Trimble SketchUp [7] is a complete and extensible 3D modelling package, widely used in 3D modelling and architectural design. CityEditor [36] is an extension for SketchUp, which supports import/export and editing of CityGML files. As an extension, CityEditor leverages the full capabilities of SketchUp, going far beyond the editing operations that the VSE supports. However, it lacks specific LoD3 tools, such as façade template application. Furthermore, CityEditor discards all semantic information when importing a CityGML file, and each surface is labeled "unclassified." Users need to assign semantic information for each surface manually. CityEditor also supports limited automatic surface classification depending on the tilt of the surface. The surface classification can then be exported to a CityGML file.

4.3.2. CityGML Importer

FME Desktop from Safe Software [35] is data conversion software which supports more than 450 formats and applications to help integrate and transform data. CityGML Importer [46] is also based on Safe Software's data transformation technology and focuses on loading and converting CityGML data; e.g., into IMX files which can be edited in InfraWorks [47] and Map 3D [48]. While loading a CityGML file, CityGML importer maintains the semantic information of each surface by storing them in separate layers. However, the hierarchical structure of the CityGML file is lost. For instance, building parts and boundary surfaces are at different hierarchical levels in the data model, but CityGML Importer discards the relationship and puts the layers on the same level. The VSE maintains the semantic information as well as the hierarchical structure of the CityGML data model.

4.3.3. Rhino City

Rhino3D [49] is a popular architectural design software which has extensive support for curves and free-form surfaces. The polygon-based editing operations of the VSE are not well suited for curved surfaces and further development is required to provide a good user experience. Through the Grasshopper extension, parametric 3D modeling can be added to Rhino3D which offers similar capabilities as the CGA based template application of the VSE. RhinoCity [50] is another extension for Rhino3D and focuses on generation of building models and supports import/export of CityGML files. RhinoCity leverage the full editing tools from Rhino and offers a far richer set of editing operations than the VSE. RhinoCity also supports automatic generation of CityGML data during the 3D model creation phase but lacks the maintenance of semantic information during further editing.

4.3.4. AutoCAD Map 3D

AutoCAD [25] is a commercial computer-aided design (CAD) and drafting software which has several domain-specific enhancements. For instance, AutoCAD Map 3D [48] is one of AutoCAD's vertical applications. AutoCAD Map 3D incorporates geographic information system and CAD data with an industry-specific toolset for GIS and 3D mapping. AutoCAD Map 3D supports importing and exporting objects into CityGML format. However, CityGML import is limited to level of detail 2 (LoD2), whereas the VSE is designed explicitly for LoD3 editing. Furthermore, much of the semantic information is lost when exporting to CityGML format from Map 3D.

4.3.5. Comparisons among VSE, CityEditor, and CityGML Importer

We compared our method with CityEditor and CityGML Importer to edit the building model shown in Figure 24. Fourteen test cases in Table 1 were chosen while other test cases were ignored, since they were not applicable in Skethup or 3ds max. For instance, test case 29 and 30 were façade template applications provided by Esri procedural runtime which were not feasible in Sketchup and 3ds max. One metric was employed: the minimum amount of mouse and keyboard interactions required for updating the semantic information of the edited geometry.

To record the metric, one user performed the 14 test cases per software. In GUI of VSE, the user can click the refresh button after an edit operation to update semantic information of the edited geometry. In CityEditor, the user need to assign semantic label manually in the context menu. CityGML Importer translates CityGML files into IMX files and stores each polygon into separate layers. The name of the layer is the semantic label of the polygon. The user can edit the name of the layer to update the semantic information. The results are shown in Table 2.

There are two observations from Table 2. First, our method is helpful for an efficient workflow with multiple newly added faces. For transformation, deletion, and deformation, no extra mouse clicks are needed to update the semantic information in Sketchup and 3ds max. For push-pull, which generates newly added faces, users need several mouse clicks to manually assign semantic labels for the newly added faces. Thanks to automatic labeling introduced in Section 3.4, manual labor is saved in updating the edited geometry. Second observation is our method has advantages in importing and exporting CityGML data models. As mentioned in Section 4.3.1, CityEditor lost all semantic information in loading the CityGML data model. The comparison excludes the number of mouse clicks in initializing the semantic label of the model in VSE, CityEditor, and CityGML Importer. Extra tools are needed to export the shape in 3ds max into CityGML format, while VSE supports the direct export of CityGML files.

Table 2. Comparisons among VSE, CityEditor, and CityGML Importer.

Test Case ID	VSE (Clicks)	CityEditor (Clicks)	CityGML Importer (Clicks)
Initialization	0	90	1
01	1	0	0
02	1	0	0
05	1	0	0
08	1	0	0
10	1	0	0
13	1	0	0
14	1	0	0
15	1	0	0
16	1	0	0
18	1	4	4
20	1	4	4
23	1	10	10
25	1	20	20
26	1	20	20

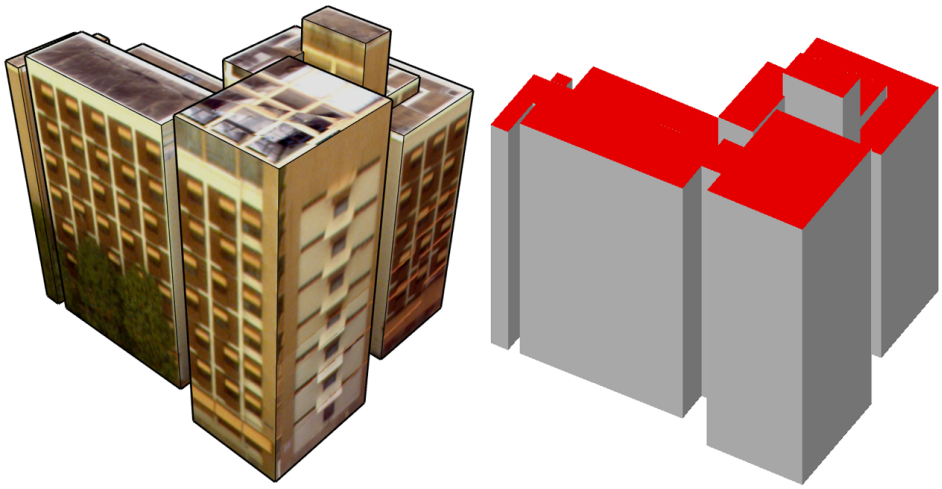

Figure 24. A building model and its semantic information for comparisons among VSE, CityEditor, and CityGML Importer.

5. Conclusions

This article introduced a data model synchronization method which preserves semantic information across editing operation. It is independent of the edit operation and depends only on geometry, UV mappings, and materials. This enables easy integration of existing and future 3D editing tools with rich data models in a broad range of 3D applications. The method was implemented in a LoD3 building editor for the Virtual Singapore project, including interactive push-pull and procedural generation of façade provided by 3rd party libraries. The quality of the method was verified with 30 common editing tasks and compared with four commercial tools.

The VSE internally uses a polygon-based representation for the geometry which is not well suited for free-form or curved surfaces. Further research is required for recovering surface parametrization and designing an easy to use toolset for editing curved surfaces.

Currently, the focus is on interactive editing operations only. It would be interesting to see how accuracy our labeling process will be in more practice cases and how the method performs in batch processing of 3D geometry, e.g., for change detection and labeling, an important topic for maintaining and updating 3D city models.

Author Contributions: Conceptualization, S.Y. and S.S.; methodology, S.S. and Z.F.; software, Esri Inc., PCL, Oracle Inc., S.S., S.Y., F.N., and X.L.; validation, X.L. and S.Y.; formal analysis, X.L. and Z.F.; investigation, X.L. and Z.F.; resources, S.Y., G.S., and X.L.; data curation, S.Y. and X.L.; writing—original draft preparation, S.Y. and X.L.; writing—review and editing, Z.T., L.M., and S.S.; visualization, X.L., S.Y., and L.M.; supervision, S.S. and G.S.; project administration, S.S. and G.S.; funding acquisition, S.S. All authors have read and agreed to the published version of the manuscript.

Funding: This material is based on research/work supported by the National Research Foundation under Virtual Singapore Award No.NRF2015VSG-AA3DCM001-024.

Acknowledgments: We would like to thank the Singapore Land Authority (SLA) for providing the LoD2 CityGML files.

Conflicts of Interest: The authors declare no conflict of interest.

Reference

1. Hagedorn, B.; Döllner, J. High-level web service for 3D building information visualization and analysis. In Proceedings of the 15th Annual ACM International Symposium on Advances in Geographic Information Systems, Seattle, WA, USA, 7–9 November 2007.
2. Psyllidis, A.; Bozzon, A.; Bocconi, S.; Bolivar, C.T. A platform for urban analytics and semantic data integration in city planning. In Proceedings of the International Conference on Computer-Aided Architectural Design Futures, Sao Paulo, Brazil, 8–10 July 2015; Springer: Berlin/Heidelberg, Germany; 2015; pp. 21–36.

3. Prandi, F.; De Amicis, R.; Piffer, S.; Soave, M.; Cadzow, S.; Boix, E.G.; D'Hondt, E. Using CityGML to deploy smart-city services for urban ecosystems. *Int. Arch. Photogramm. Remote Sens. Spat. Inf. Sci.* **2013**, *4*, W1. [CrossRef]
4. Volpi, M.; Ferrari, V. Structured prediction for urban scene semantic segmentation with geographic context. In Proceedings of the 2015 Joint Urban Remote Sensing Event (JURSE), Lausanne, Switzerland, 30 March–1 April 2015; pp. 1–4.
5. Kagal, L. A Policy-Based Approach to Governing Autonomous Behavior in Distributed Environments. Ph.D. Thesis, University of Maryland Baltimore County, Baltimore, MD, USA, 2004.
6. Lipp, M.; Wonka, P.; Müller, P. PushPull++. *ACM Trans. Graph. (TOG)* **2014**, *33*, 130. [CrossRef]
7. TRIMBLE. Sketchup. Available online: www.sketchup.com (accessed on 19 November 2019).
8. Parish, Y.I.; Müller, P. Procedural modeling of cities. In Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques, Los Angeles, CA, USA, 12–17 August 2001; pp. 301–308.
9. Müller, P.; Wonka, P.; Haegler, S.; Ulmer, A.; Van Gool, L. Procedural modeling of buildings. *ACM Trans. Graph. (TOG)* **2006**, *25*, 614–623. [CrossRef]
10. Esri. Procedural Runtime Whitepaper. Available online: https://esri.github.io/esri-cityengine-sdk/html/index.html (accessed on 19 November 2019).
11. Alegre, F.; Dellaert, F. A probabilistic approach to the semantic interpretation of building facades. In Proceedings of the CIPA International Workshop on Vision Techniques Applied to the Rehabilitation of City Centres, Lisbonne, Portugal, 25–27 October 2004.
12. Verdie, Y.; Lafarge, F.; Alliez, P. Lod generation for urban scenes. *ACM Trans. Graph.* **2015**, *34*, 30. [CrossRef]
13. Zhu, Q.; Li, Y.; Hu, H.; Wu, B. Robust point cloud classification based on multi-level semantic relationships for urban scenes. *ISPRS J. Photogramm. Remote Sens.* **2017**, *129*, 86–102. [CrossRef]
14. Wu, C.; Lenz, I.; Saxena, A. Hierarchical Semantic Labeling for Task-Relevant RGB-D Perception. In Proceedings of the 2014 Robotics: Science and Systems Conference, Berkeley, CA, USA, 12–16 July 2014.
15. Rook, M.; Biljecki, F.; Diakité, A. Towards Automatic Semantic Labelling of 3D City Models. *ISPRS Ann. Photogramm. Remote Sens. Spat. Inf. Sci.* **2016**, *4*. [CrossRef]
16. Sundar, H.; Silver, D.; Gagvani, N.; Dickinson, S. Skeleton based shape matching and retrieval. In Proceedings of the 2003 Shape Modeling International, Seoul, Korea, 12–15 May 2003; pp. 130–139.
17. Xie, J.; Dai, G.; Zhu, F.; Wong, E.K.; Fang, Y. Deepshape: Deep-learned shape descriptor for 3D shape retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *39*, 1335–1345. [CrossRef] [PubMed]
18. Besl, P.J.; McKay, N.D. Method for registration of 3-D shapes. Sensor fusion IV: Control paradigms and data structures. *Int. Soc. Opt. Photonics* **1992**, *1611*, 586–606.
19. Zhang, Z. Iterative point matching for registration of free-form curves and surfaces. *Int. J. Comput. Vis.* **1994**, *13*, 119–152. [CrossRef]
20. Rusinkiewicz, S.; Levoy, M. Efficient variants of the ICP algorithm. In Proceedings of the 3rd International Conference on 3D Digital Imaging and Modeling (3DIM 2001), Quebec City, QC, Canada, 28 May–1 June 2001; Volume 1, pp. 145–152.
21. Fitzgibbon, A.W. Robust registration of 2D and 3D point sets. *Image Vis. Comput.* **2003**, *21*, 1145–1153. [CrossRef]
22. Myronenko, A.; Song, X. Point set registration: Coherent point drift. *IEEE Trans. Pattern Anal. Mach. Intell.* **2010**, *32*, 2262–2275. [CrossRef] [PubMed]
23. Ma, J.; Zhao, J.; Tian, J.; Yuille, A.L.; Tu, Z. Robust point matching via vector field consensus. *IEEE Trans. Image Process.* **2014**, *23*, 1706–1721. [CrossRef] [PubMed]
24. Ge, S.; Fan, G.; Ding, M. Non-rigid point set registration with global-local topology preservation. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops, Columbus, OH, USA, 23–28 June 2014; pp. 245–251.
25. AUTODESK. Autocad. Available online: http://www.autodesk.com/autocad (accessed on 19 November 2019).
26. AUTODESK. Maya. Available online: http://www.autodesk.com/maya (accessed on 19 November 2019).
27. Esri. CityEngine. Available online: https://www.esri.com/en-us/arcgis/products/esri-cityengine/overview (accessed on 10 November 2019).
28. Esri. ArcGIS Pro. Available online: https://www.esri.com/zh-cn/arcgis/products/arcgis-pro/resources (accessed on 10 November 2019).

29. Biljecki, F.; Ledoux, H.; Stoter, J.; Zhao, J. Formalisation of the level of detail in 3D city modelling. *Comput. Environ. Urban Syst.* **2014**, *48*, 1–15. [CrossRef]
30. Kolbe, T.H.; Gröger, G.; Plümer, L. CityGML: Interoperable access to 3D city models. In *Geo-Information for Disaster Management*; Springer: Berlin, Germany, 2005; pp. 883–899.
31. BentleySystems. Bentley Map. Available online: https://www.bentley.com/en/products/product-line/asset-performance/opencities-map (accessed on 19 November 2019).
32. Bitmanagement. BS Contact Geo. 2019. Available online: http://www.bitmanagement.de/en/products/interactive-3d-clients/bs-contact-geo (accessed on 19 November 2019).
33. Fraunhofer Institute for Computer Graphics Research(IGD) CityServer3D. 2019. Available online: http://www.cityserver3d.de/en/ (accessed on 19 November 2019).
34. Synthesis, C. CodeSynthesis XSD. 2019. Available online: https://www.codesynthesis.com/ (accessed on 19 November 2019).
35. SAFESOOFTWARE. FME. Available online: https://www.safe.com/fme/ (accessed on 10 August 2019).
36. 3Dis. CityEditor. Available online: https://www.3dis.de/cityeditor/ (accessed on 19 November 2019).
37. UVMSystems. CityGrid. Available online: http://www.uvmsystems.com/index.php/en/ (accessed on 19 November 2019).
38. Foley, J.D.; Van, F.D.; Van Dam, A.; Feiner, S.K.; Hughes, J.F.; Hughes, J.; Angel, E. *Computer Graphics: Principles and Practice*; Addison-Wesley Professional: Boston, MA, USA, 1996; Volume 12110.
39. Birdal, T.; Ilic, S. A point sampling algorithm for 3d matching of irregular geometries. In Proceedings of the 2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Vancouver, BC, Canada, 24–28 September 2017; pp. 6871–6878.
40. Rodolà, E.; Albarelli, A.; Cremers, D.; Torsello, A. A simple and effective relevance-based point sampling for 3D shapes. *Pattern Recognit. Lett.* **2015**, *59*, 41–47. [CrossRef]
41. DeRose, T.; Meyer, M.; Bakshi, S. Mesh Transfer Using UV-Space. US Patent 8,482,569, 9 July 2013.
42. Gröger, G.; Kolbe, T.H.; Nagel, C.; Häfele, K.H. *OGC City Geography Markup Language (CityGML) Encoding Standard*; Open Geospatial Consortium: Wayland, MA, USA, 2012.
43. Building and Construction Authority. Code of Practice on Buildable Design. Available online: https://www.bca.gov.sg/BuildableDesign/others/copbddec00.pdf (accessed on 19 November 2019).
44. PCL. Point Cloud Library. Available online: http://pointclouds.org/contact.html (accessed on 19 November 2019).
45. KIT. FZKViewer. Available online: https://www.iai.kit.edu/1302.php (accessed on 10 August 2019).
46. SafeSoftware. CityGML Importer. Available online: https://www.safe.com/citygml-importer (accessed on 19 November 2019).
47. Autodesk. InfraWorks. Available online: https://www.autodesk.com/products/infraworks/overview (accessed on 19 November 2019).
48. AUTODESK. AutoCAD Map 3D. Available online: https://knowledge.autodesk.com/support/autocad-map-3d (accessed on 19 November 2019).
49. Associates, R.M. Rhino 3D. Available online: https://www.rhino3d.com/ (accessed on 19 November 2019).
50. RhinoTerrain. RhinoCity. 2019. Available online: https://www.rhinoterrain.com/en/rhinocity.html (accessed on 19 November 2019).